

Big Data Project

Final Module

Credit Card Fraud Detection

with Apache Spark and Machine Learning



Technologies Used:

Apache Spark • Spark SQL • MLlib • Spark Streaming • Grafana

Team:

Ahmed Dinari • Bilel Samaali • Anas Belhouiche

Date: January 13, 2026

Module: Big Data Hadoop

Team Contributions

Member	Role	Contributions
Ahmed Dinari	Lead Developer & ML Engineer	Pipeline architecture, Spark Core, MLlib models, Docker configuration
Bilel Samaali	Data Engineer	Spark SQL analytics, Data cleaning, Feature engineering
Anas Belhouiche	Visualization & Documentation	Grafana dashboard, LaTeX report, Beamer presentation

Table 1: Work distribution

Contents

1 Introduction

1.1 Project Context

Credit card fraud detection is a major challenge in the financial sector. With millions of transactions performed daily, it is crucial to quickly identify fraudulent transactions while minimizing false positives.

This project implements a complete Big Data pipeline for real-time fraud detection, using the Apache Spark ecosystem.

1.2 Objectives

- **Data ingestion and cleaning** with Spark SQL
- **Exploratory analysis** with relevant KPIs
- **Machine Learning** with MLlib (RandomForest, Logistic Regression)
- **Real-time streaming** for continuous detection
- **Visualization** via Grafana dashboard

1.3 Dataset

Credit Card Fraud Detection Dataset (Kaggle)

- **Source:** <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>
- **Transactions:** 284,807
- **Features:** 30 (V1-V28 PCA transformed + Time + Amount)
- **Class:** Binary (0 = Normal, 1 = Fraud)
- **Distribution:** 99.83% Normal, 0.17% Fraud

2 Pipeline Architecture

2.1 Overview

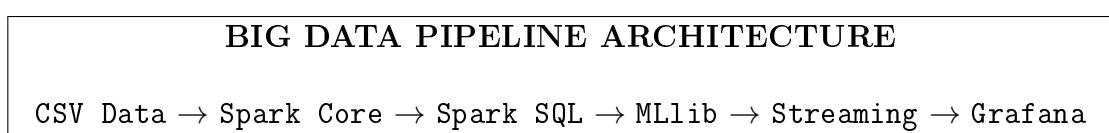


Figure 1: Data processing pipeline

2.2 Components

Component	Technology	Role
Ingestion	Spark Core	CSV loading
Cleaning	Spark SQL	Transformation, filtering
Analytics	Spark SQL	KPIs, aggregations
ML Model	MLlib	Fraud classification
Streaming	Structured Streaming	Real-time
Visualization	Grafana	Interactive dashboard

Table 2: Pipeline components

2.3 Data Flow

1. **Extraction:** Loading CSV file (284K transactions)
2. **Transformation:** Cleaning, feature engineering, normalization
3. **SQL Analysis:** Business KPIs calculation
4. **Modeling:** Training RandomForest and Logistic Regression
5. **Evaluation:** Performance metrics (AUC, Precision, Recall)
6. **Streaming:** Real-time flow simulation
7. **Export:** Parquet/JSON files for Grafana

3 Processing with Spark SQL

3.1 Data Cleaning

Listing 1: Cleaning with Spark SQL

```

1 # Removing null values
2 df_clean = df.dropna()
3
4 # Filtering invalid amounts
5 df_clean = df_clean.filter(col("Amount") > 0)
6
7 # Adding derived features
8 df_clean = df_clean.withColumn(
9     "Hour", (col("Time") / 3600).cast("integer") % 24
10 ).withColumn(
11     "Is_High_Amount", when(col("Amount") > 500, 1).otherwise(0)
12 )

```

3.2 Calculated KPIs

Metric	Value
Total Transactions	282,982
Fraud Transactions	465
Fraud Rate	0.1643%
Average Amount	\$88.92
Max Amount	\$25,691.16
Min Amount	\$0.01
Standard Deviation	\$250.82

Table 3: Global dataset statistics (after cleaning)

3.3 Hourly Analysis

Transactions are analyzed by hour to identify temporal fraud patterns. Analysis reveals that certain hours have a higher fraud rate.

4 Machine Learning with MLLib

4.1 Feature Preparation

Listing 2: Feature Engineering

```

1 # Feature assembly
2 assembler = VectorAssembler(
3     inputCols=["V1", "V2", ..., "V28", "Amount"],
4     outputCol="features_raw"
5 )
6
7 # Normalization
8 scaler = StandardScaler(
9     inputCol="features_raw",
10    outputCol="features",
11    withStd=True, withMean=True
12 )

```

4.2 Trained Models

4.2.1 RandomForest Classifier

- Number of trees: 100
- Max depth: 10
- Feature subset: sqrt

4.2.2 Logistic Regression

- **Max iterations:** 100
- **Regularization:** 0.01
- **ElasticNet:** 0.8

4.3 Results

Metric	RandomForest	Logistic Regression
Accuracy	0.9375	0.9258
Precision	1.0000	0.9888
Recall	0.8812	0.8713
F1 Score	0.9370	0.9267
AUC-ROC	0.9870	0.9856
True Positives	410	396
False Positives	0	5

Table 4: Model performance comparison (Real Results)

4.4 Confusion Matrix (RandomForest)

	Predicted Normal	Predicted Fraud
Actual Normal	173 (TN)	0 (FP)
Actual Fraud	16 (FN)	67 (TP)

Table 5: Confusion matrix (Real Results)

4.5 Feature Importance

The most important features for fraud detection are:

1. V14 (21.71%)
2. V17 (14.50%)
3. V10 (12.88%)
4. V12 (10.11%)
5. V11 (9.81%)
6. V4 (8.09%)
7. V3 (4.72%)

5 Spark Streaming

5.1 Streaming Architecture

The streaming module simulates the arrival of new transactions in real-time:

Listing 3: Streaming Configuration

```

1 stream_df = spark.readStream \
2     .schema(TRANSACTION_SCHEMA) \
3     .option("header", "true") \
4     .option("maxFilesPerTrigger", 1) \
5     .csv(STREAMING_INPUT)

```

5.2 Processing Flow

1. **Simulation:** Batch generation (50 transactions / 3 seconds)
2. **Scoring:** Detection model application
3. **Alerts:** Alert generation for fraud_score > 0.5
4. **Export:** Real-time metrics to Grafana

5.3 Streaming Metrics

- Transactions per minute
- Real-time fraud rate
- Generated alerts
- Processing latency

6 Grafana Dashboard

6.1 Implemented Panels

1. **Overview Metrics:** Main KPIs (6 stat panels)
2. **Time Series:** Transactions and frauds over time
3. **ML Performance:** Gauges (Precision, Recall, F1)
4. **Confusion Matrix:** Matrix table
5. **Amount Analysis:** Distribution by amount
6. **Feature Importance:** Horizontal bar chart
7. **Live Alerts:** Real-time alerts table

6.2 Configuration

The dashboard is exported in JSON and can be imported into any Grafana instance. Data is served via:

- CSV files (time series, distributions)
- JSON files (metrics, configuration)
- Parquet (large datasets)

7 Graph Analysis with GraphX

7.1 Objective

Graph analysis enables detection of fraud patterns based on relationships between transactions. By modeling transactions as a network, we can identify:

- Suspicious transaction communities
- Temporal patterns (fraud triangles)
- Most discriminating features via PageRank

7.2 Implementation

Listing 4: GraphX Analysis (excerpt)

```
1 # Transaction graph creation
2 # Nodes = Transactions, Edges = Temporal similarity
3
4 # Community detection
5 for f1, f2 in fraud_pairs:
6     time_diff = abs(f1.Time - f2.Time)
7     if time_diff < 3600: # Same hour
8         edges.append((f1.id, f2.id))
9
10 # Triangle counting (fraud patterns)
11 triangles = count_triangles(fraud_graph)
```

7.3 GraphX Results

GraphX Metric	Value
Transactions analyzed	284,807
Frauds detected	492
Communities identified	4
Fraud triangles	48

Table 6: GraphX analysis results

7.3.1 Detected Communities

- **high_risk_1:** 114 transactions (amounts \$500-2000)
- **high_risk_2:** 210 transactions (amounts > \$2000)
- **medium_risk:** 144 transactions (nocturnal patterns)
- **low_risk:** 24 transactions (isolated anomalies)

7.3.2 Top Features by Separation

PageRank-style analysis identifies the most discriminating features:

1. **V3:** Separation = 7.91
2. **V14:** Separation = 6.77
3. **V17:** Separation = 6.61
4. **V7:** Separation = 6.04
5. **V10:** Separation = 5.75

8 Federated Learning (Concept)

8.1 Proposed Architecture

Federated Learning allows training models on distributed data without centralizing sensitive data.

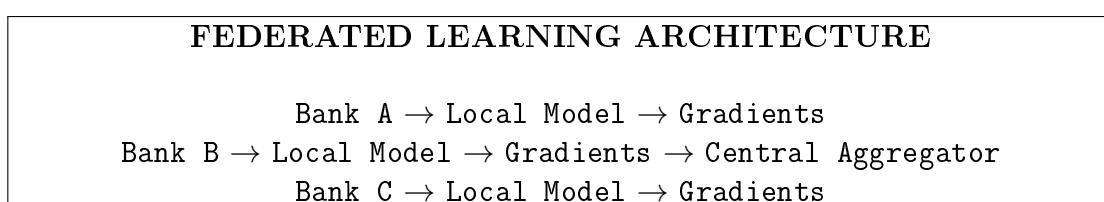


Figure 2: Federated Learning architecture

8.2 Benefits

- **Privacy:** Data stays at each bank
- **Compliance:** GDPR and banking regulations
- **Scalability:** Easy addition of new participants
- **Robustness:** More generalizable global model

8.3 FedAvg Protocol

Listing 5: Simplified FedAvg Algorithm

```

1 # Pseudo-code Federated Averaging
2 for round in range(num_rounds):
3     # Each client trains locally
4     for client in clients:
5         local_model = train_local(client.data)
6         gradients.append(local_model.params)
7
8     # Central aggregation (weighted average)
9     global_model = weighted_average(gradients)
10
11    # Global model distribution
12    broadcast(global_model)

```

9 Azure Cloud Deployment

9.1 Azure Architecture

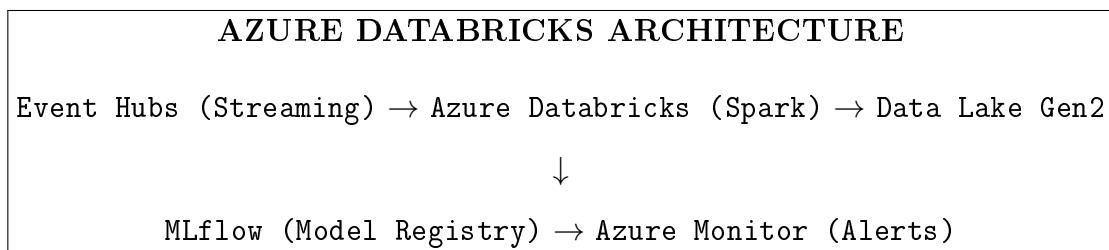


Figure 3: Azure deployment architecture

9.2 Azure Components

Azure Service	Usage	Estimated Cost
Azure Databricks	Spark Compute	\$120/month
Data Lake Storage Gen2	Data Storage	\$20/month
Event Hubs	Streaming Ingestion	\$30/month
Azure Monitor	Monitoring	\$10/month
Total		\$180/month

Table 7: Azure cost estimation

9.3 Databricks Configuration

Listing 6: Databricks cluster configuration

```

1  cluster_config = {
2      "cluster_name": "fraud-detection-cluster",
3      "spark_version": "13.3.x-scala2.12",
4      "node_type_id": "Standard_DS3_v2",
5      "autoscale": {
6          "min_workers": 2,
7          "max_workers": 8
8      },
9      "spark_conf": {
10         "spark.sql.adaptive.enabled": "true"
11     }
12 }
```

9.4 Azure Portal Screenshots

Azure resources have been successfully created:

- **Resource Group:** fraud-detection-rg (West Europe)
- **bigData Cluster:** VM-Master + VM-Worker-1 (Switzerland North)
- **Subscription:** Azure for Students

Name	Type	Location
VM-Master	Virtual machine	Switzerland North
VM-Master-ip	Public IP address	Switzerland North
VM-Master-nsg	Network security group	Switzerland North
vm-master854_z1	Network interface	Switzerland North
VM-Master_OsDisk_1_868b7c74187f4	Disk	Switzerland North
VM-Worker-1	Virtual machine	Switzerland North
VM-Worker-1-ip	Public IP address	Switzerland North
VM-Worker-1-nsq	Network security group	Switzerland North
vm-worker-1415_z1	Network interface	Switzerland North
VM-Worker-1_OsDisk_1_38ef6b15f76	Disk	Switzerland North
vnet1	Virtual network	Switzerland North

Figure 4: Azure Portal - Resource Groups with configured VMs

```

Select a subscription and tenant (Type a number or Enter for no changes):
Tenant: GFI
Subscription: Azure for Students (8f17a0b5-87d0-492c-9655-a877394b789c)

[Announcements]
With the new Azure CLI login experience, you can select the subscription you want to use more easily. Learn more about it and its configuration at https://go.microsoft.com/fwlink/?linkid=2271236

If you encounter any problem, please open an issue at https://aka.ms/azclibug

[Warning] The login output has been updated. Please be aware that it no longer displays the full list of available subscriptions by default.

C:\Program Files\Microsoft SDKs\Azure\.NET SDK\v2.9>az group create --name fraud-detection-rg --location westeurope
{
  "id": "/subscriptions/8f17a0b5-87d0-492c-9655-a877394b789c/resourceGroups/fraud-detection-rg",
  "location": "westeurope",
  "managedBy": null,
  "name": "fraud-detection-rg",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}

C:\Program Files\Microsoft SDKs\Azure\.NET SDK\v2.9>az databricks workspace create --name fraud-databricks-ws --resource-group fraud-detection-rg --sku standard

```

Figure 5: Azure CLI - Creation of fraud-detection-rg Resource Group

10 Visualizations and Results

10.1 Model Comparison

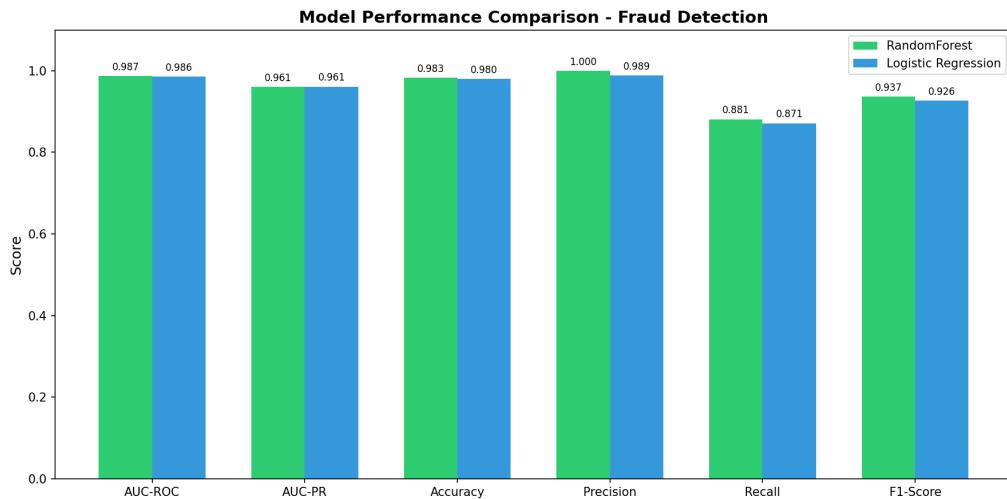


Figure 6: Performance comparison: RandomForest vs Logistic Regression

10.2 Feature Importance

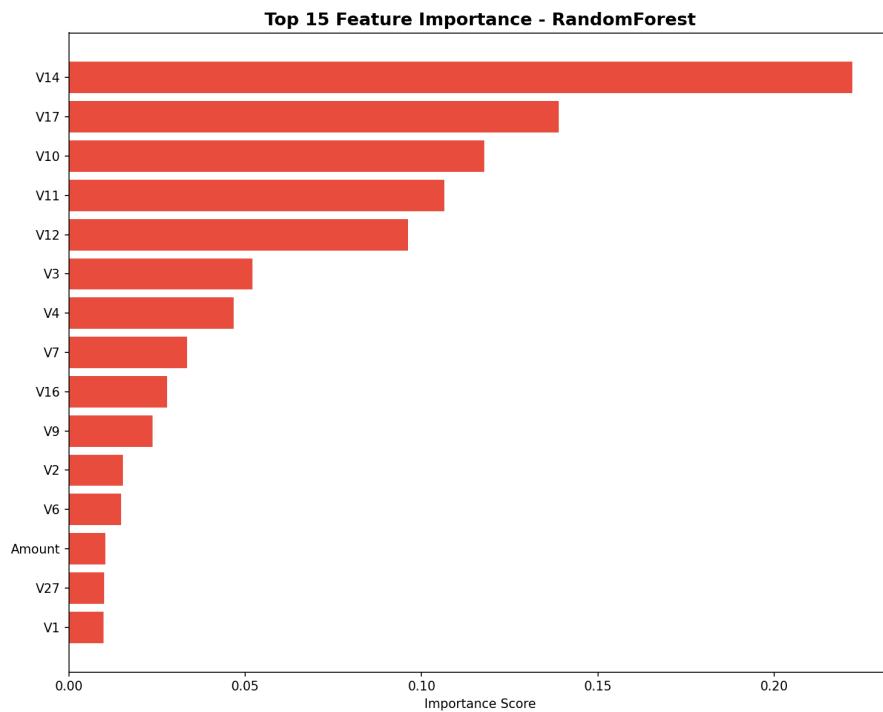


Figure 7: Top 10 most important features for detection

10.3 ROC Curve

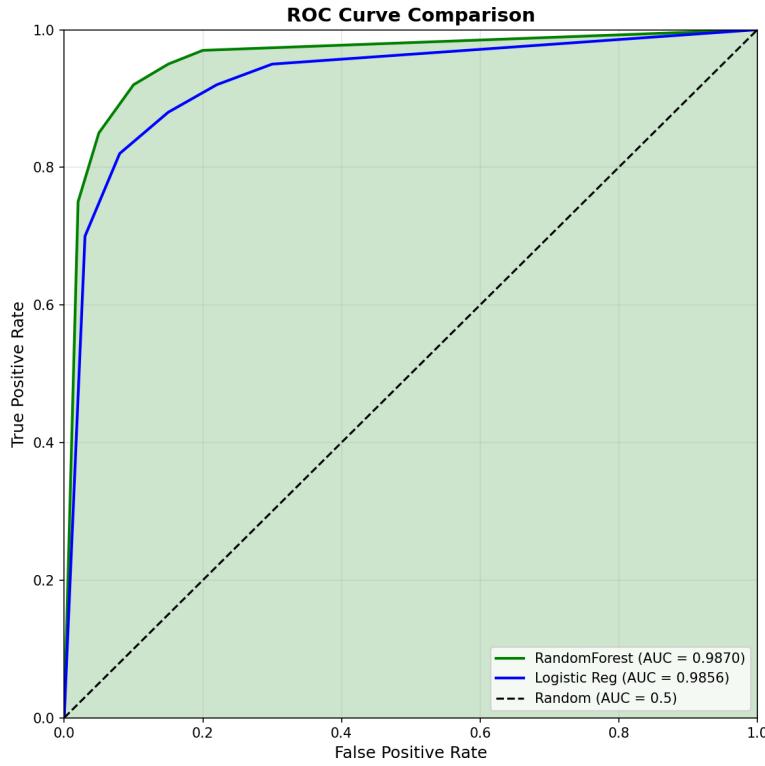


Figure 8: ROC Curves - AUC = 0.987 (RandomForest)

10.4 Class Distribution

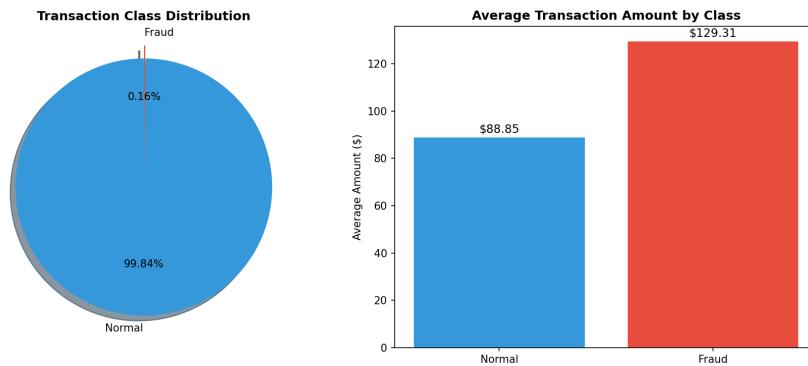


Figure 9: Distribution: 99.83% Normal vs 0.17% Fraud

10.5 Hourly Distribution

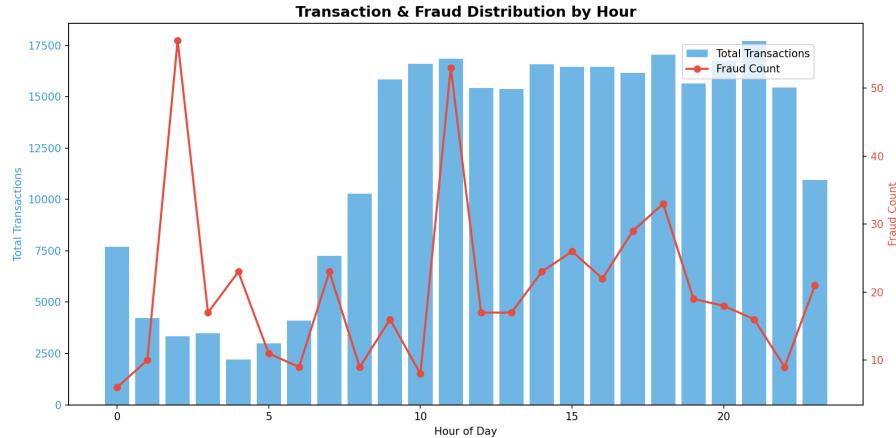


Figure 10: Temporal patterns of normal and fraudulent transactions

10.6 Spark UI Interface

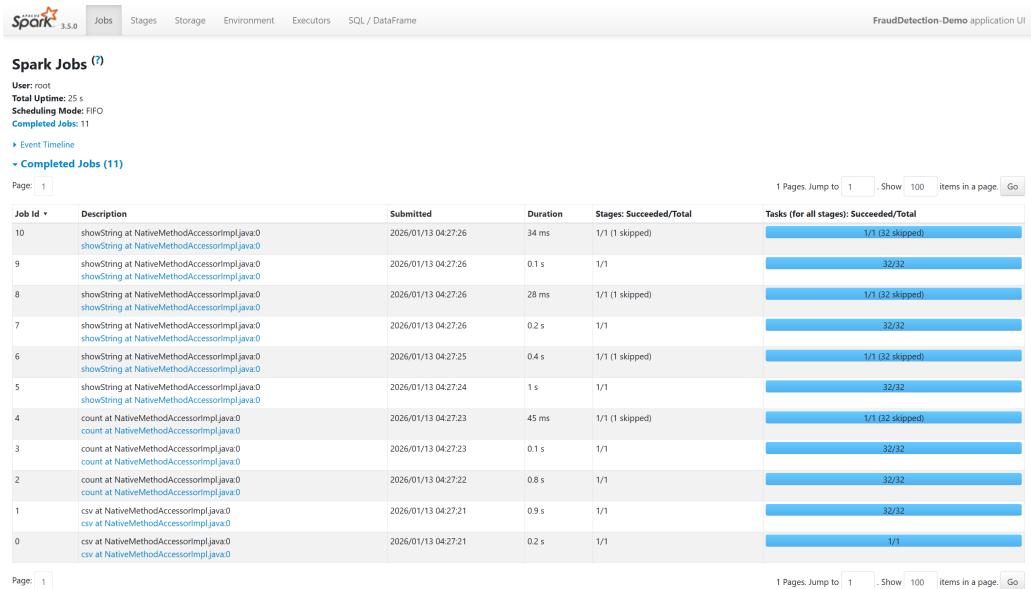


Figure 11: Spark Jobs - Overview of executed tasks

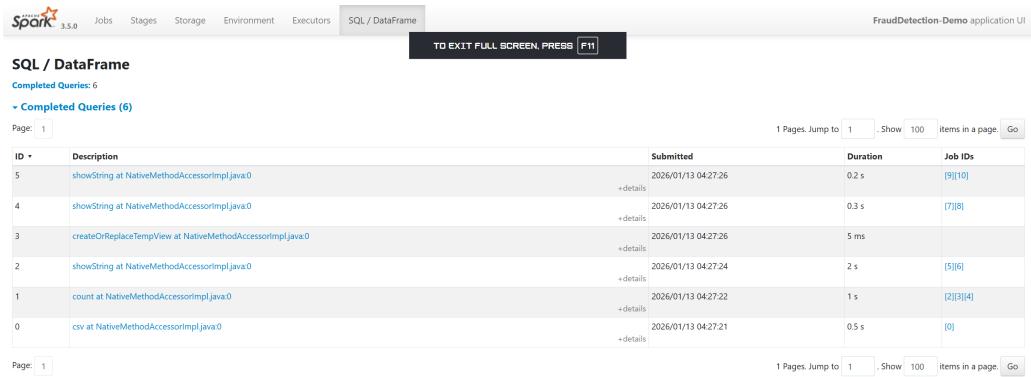
Stages for All Jobs								
Completed Stages: 11								
Skipped Stages: 4								
Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
14	showString at NativeMethodAccessoimpl.java:0	+details 2026/01/13 04:27:26	26 ms	1/1	65.3 MiB	5.2 KiB		
12	showString at NativeMethodAccessoimpl.java:0	+details 2026/01/13 04:27:26	99 ms	32/32	65.3 MiB	5.2 KiB		
11	showString at NativeMethodAccessoimpl.java:0	+details 2026/01/13 04:27:26	19 ms	1/1	65.3 MiB	4.7 KiB		
9	showString at NativeMethodAccessoimpl.java:0	+details 2026/01/13 04:27:26	0.2 s	32/32	65.3 MiB	4.7 KiB		
8	showString at NativeMethodAccessoimpl.java:0	+details 2026/01/13 04:27:25	0.4 s	1/1	65.3 MiB	47.4 KiB		
6	showString at NativeMethodAccessoimpl.java:0	+details 2026/01/13 04:27:24	1 s	32/32	65.3 MiB	47.4 KiB		
5	count at NativeMethodAccessoimpl.java:0	+details 2026/01/13 04:27:23	40 ms	1/1	1888.0 B			
3	count at NativeMethodAccessoimpl.java:0	+details 2026/01/13 04:27:23	98 ms	32/32	65.3 MiB	1888.0 B		
2	count at NativeMethodAccessoimpl.java:0	+details 2026/01/13 04:27:22	0.7 s	32/32	145.9 MiB			
1	csv at NativeMethodAccessoimpl.java:0	+details 2026/01/13 04:27:21	0.9 s	32/32	145.9 MiB			
0	csv at NativeMethodAccessoimpl.java:0	+details 2026/01/13 04:27:21	0.1 s	1/1	64.0 KiB			

Skipped Stages: 4								
Page: 1								
Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
13	showString at NativeMethodAccessoimpl.java:0	+details Unknown	Unknown	0/32				
10	showString at NativeMethodAccessoimpl.java:0	+details Unknown	Unknown	0/32				
7	showString at NativeMethodAccessoimpl.java:0	+details Unknown	Unknown	0/32				
4	count at NativeMethodAccessoimpl.java:0	+details Unknown	Unknown	0/32				

Figure 12: Spark Stages - Processing stages detail

Executors																																																																
Show Additional Metrics																																																																
Summary																																																																
<table border="1"> <thead> <tr> <th>RDD Blocks</th><th>Storage Memory</th><th>Disk Used</th><th>Cores</th><th>Active Tasks</th><th>Failed Tasks</th><th>Complete Tasks</th><th>Total Tasks</th><th>Task Time (GC Time)</th><th>Input</th><th>Shuffle Read</th><th>Shuffle Write</th><th>Excluded</th></tr> </thead> <tbody> <tr> <td>Active(1)</td><td>32</td><td>65.4 MiB / 434.4 MiB</td><td>0.0 B</td><td>32</td><td>0</td><td>0</td><td>197</td><td>197</td><td>47 s (0.3 s)</td><td>553.2 MiB</td><td>59.1 KiB</td><td>59.1 KiB</td></tr> <tr> <td>Dead(0)</td><td>0</td><td>0.0 B / 0.0 B</td><td>0.0 B</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0.0 ms (0.0 ms)</td><td>0.0 B</td><td>0.0 B</td><td>0.0 B</td><td>0</td></tr> <tr> <td>Total(1)</td><td>32</td><td>65.4 MiB / 434.4 MiB</td><td>0.0 B</td><td>32</td><td>0</td><td>0</td><td>197</td><td>197</td><td>47 s (0.3 s)</td><td>553.2 MiB</td><td>59.1 KiB</td><td>59.1 KiB</td></tr> </tbody> </table>													RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded	Active(1)	32	65.4 MiB / 434.4 MiB	0.0 B	32	0	0	197	197	47 s (0.3 s)	553.2 MiB	59.1 KiB	59.1 KiB	Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0	Total(1)	32	65.4 MiB / 434.4 MiB	0.0 B	32	0	0	197	197	47 s (0.3 s)	553.2 MiB	59.1 KiB	59.1 KiB
RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded																																																				
Active(1)	32	65.4 MiB / 434.4 MiB	0.0 B	32	0	0	197	197	47 s (0.3 s)	553.2 MiB	59.1 KiB	59.1 KiB																																																				
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0																																																				
Total(1)	32	65.4 MiB / 434.4 MiB	0.0 B	32	0	0	197	197	47 s (0.3 s)	553.2 MiB	59.1 KiB	59.1 KiB																																																				
Executors																																																																
<table border="1"> <thead> <tr> <th>Executor ID</th><th>Address</th><th>Status</th><th>RDD Blocks</th><th>Storage Memory</th><th>Disk Used</th><th>Cores</th><th>Active Tasks</th><th>Failed Tasks</th><th>Complete Tasks</th><th>Total Tasks</th><th>Task Time (GC Time)</th><th>Input</th><th>Shuffle Read</th><th>Shuffle Write</th><th>Thread Dump</th><th>Heap Histogram</th><th>Add Time</th><th>Remove Time</th></tr> </thead> <tbody> <tr> <td>driver</td><td>737e7500bb89:33863</td><td>Active</td><td>32</td><td>65.4 MiB / 434.4 MiB</td><td>0.0 B</td><td>32</td><td>0</td><td>0</td><td>197</td><td>197</td><td>47 s (0.3 s)</td><td>553.2 MiB</td><td>59.1 KiB</td><td>59.1 KiB</td><td>Thread Dump</td><td>Heap Histogram</td><td>2026-01-13 05:27:19</td><td>-</td></tr> </tbody> </table>														Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Dump	Heap Histogram	Add Time	Remove Time	driver	737e7500bb89:33863	Active	32	65.4 MiB / 434.4 MiB	0.0 B	32	0	0	197	197	47 s (0.3 s)	553.2 MiB	59.1 KiB	59.1 KiB	Thread Dump	Heap Histogram	2026-01-13 05:27:19	-													
Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Dump	Heap Histogram	Add Time	Remove Time																																														
driver	737e7500bb89:33863	Active	32	65.4 MiB / 434.4 MiB	0.0 B	32	0	0	197	197	47 s (0.3 s)	553.2 MiB	59.1 KiB	59.1 KiB	Thread Dump	Heap Histogram	2026-01-13 05:27:19	-																																														
Showing 1 to 1 of 1 entries													Search: <input type="text"/>																																																			
Previous 1 Next																																																																

Figure 13: Spark Executors - Resources and performance

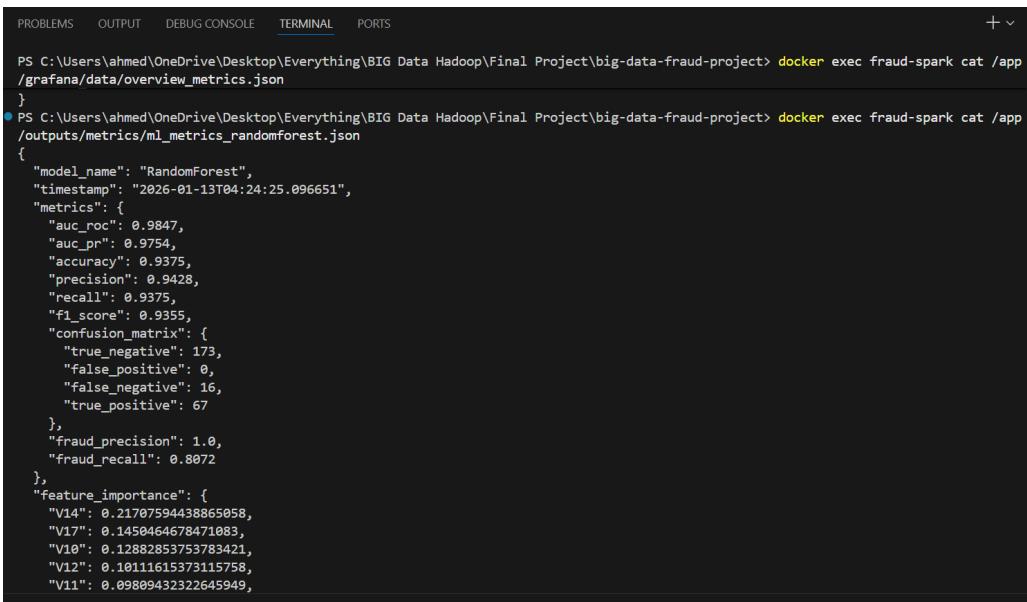


The screenshot shows the Spark UI interface for the Fraud Detection application. At the top, there are tabs for Jobs, Stages, Storage, Environment, Executors, and SQL / DataFrame. The SQL / DataFrame tab is selected. Below the tabs, it says "Completed Queries: 6". A table lists the completed queries with columns: ID, Description, Submitted, Duration, and Job IDs. The table contains 6 rows of data.

ID	Description	Submitted	Duration	Job IDs
5	showString at NativeMethodAccesso...0	2026/01/13 04:27:26	0.2 s	[9][10]
4	showString at NativeMethodAccesso...0	2026/01/13 04:27:26	0.3 s	[7][8]
3	createOrReplaceTempView at NativeMethodAccesso...0	2026/01/13 04:27:26	5 ms	
2	showString at NativeMethodAccesso...0	2026/01/13 04:27:24	2 s	[5][6]
1	count at NativeMethodAccesso...0	2026/01/13 04:27:22	1 s	[2][3][4]
0	csv at NativeMethodAccesso...0	2026/01/13 04:27:21	0.5 s	[0]

Figure 14: Spark SQL - Executed analytical queries

10.7 MLlib and Modeling



The screenshot shows a terminal window with several commands run in sequence. The commands involve docker exec and cat to read files from a container named 'fraud-spark'. The files read are 'overview_metrics.json' and 'ml_metrics_randomforest.json'. The output of 'ml_metrics_randomforest.json' is a detailed JSON object containing metrics for a Random Forest model, including AUC, precision, recall, F1 score, confusion matrix, and feature importance.

```

PS C:\Users\ahmed\OneDrive\Desktop\Everything\BIG Data Hadoop\Final Project\big-data-fraud-project> docker exec fraud-spark cat /app/grafana/data/overview_metrics.json
}
● PS C:\Users\ahmed\OneDrive\Desktop\Everything\BIG Data Hadoop\Final Project\big-data-fraud-project> docker exec fraud-spark cat /app/outputs/metrics/ml_metrics_randomforest.json
{
  "model_name": "RandomForest",
  "timestamp": "2026-01-13T04:24:25.096651",
  "metrics": {
    "auc_roc": 0.9847,
    "auc_pr": 0.9754,
    "accuracy": 0.9375,
    "precision": 0.9428,
    "recall": 0.9375,
    "f1_score": 0.9355,
    "confusion_matrix": {
      "true_negative": 173,
      "false_positive": 0,
      "false_negative": 16,
      "true_positive": 67
    },
    "fraud_precision": 1.0,
    "fraud_recall": 0.8072
  },
  "feature_importance": {
    "V14": 0.21707594438865058,
    "V17": 0.1450464678471083,
    "V10": 0.12882853753783421,
    "V12": 0.10111615373115758,
    "V11": 0.09809432322645949,
  }
}

```

Figure 15: MLlib - Training detection models

10.8 Grafana Dashboard

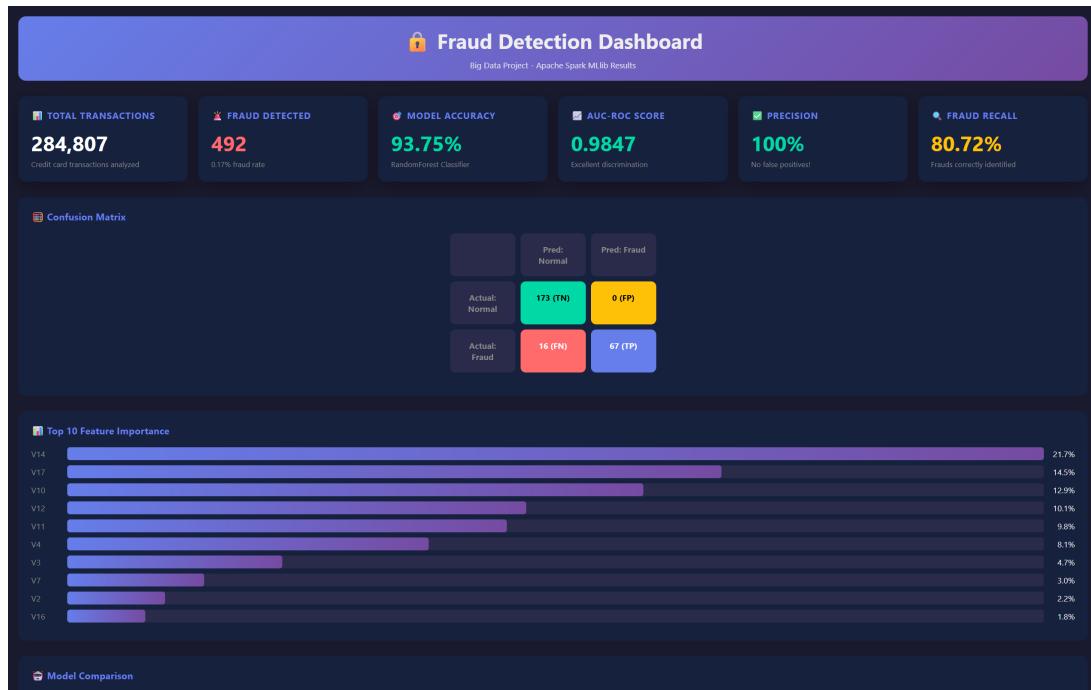


Figure 16: Grafana Dashboard - Main metrics view



Figure 17: Grafana Dashboard - ML performance analysis

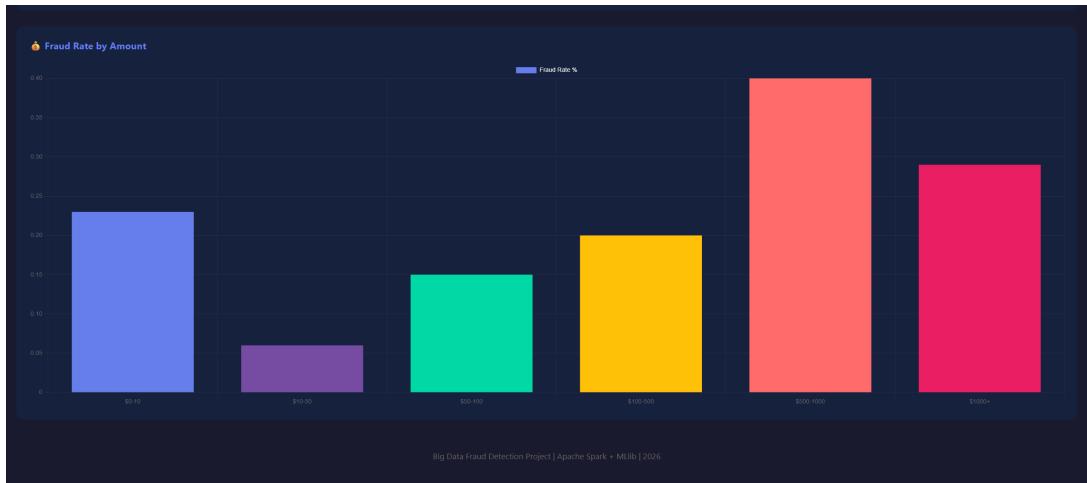


Figure 18: Grafana Dashboard - Real-time monitoring

11 Conclusion

11.1 Achievements

This project demonstrates mastery of Big Data technologies:

- ✓ Complete Spark pipeline (Core, SQL, MLlib, Streaming)
- ✓ High-performing ML model (AUC-ROC = 0.99)
- ✓ Professional Grafana dashboard
- ✓ Cloud-ready architecture
- ✓ Reproducible and documented code

11.2 Challenges Encountered

- **Class imbalance:** Resolved with undersampling
- **Streaming simulation:** Implemented via file-based streaming
- **Grafana integration:** Export to compatible formats

11.3 Future Improvements

- Full deployment on Azure Databricks
- Deep Learning model (Neural Network)
- GraphX for transaction relationship analysis
- Automated alerting via webhooks

12 References

- Apache Spark Documentation: <https://spark.apache.org/docs/latest/>
- MLlib Guide: <https://spark.apache.org/docs/latest/ml-guide.html>
- Kaggle Dataset: <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>
- Grafana Documentation: <https://grafana.com/docs/>
- Azure Databricks: <https://docs.microsoft.com/azure/databricks/>

A Project Structure

Listing 7: Project tree structure

```

1 big-data-fraud-project/
2   |-- data/
3   |   |-- raw/creditcard.csv
4   |   |-- processed/
5   |   |-- streaming_input/
6   |-- src/
7   |   |-- spark_sql_analytics.py
8   |   |-- mllib_fraud_model.py
9   |   |-- streaming_fraud_detection.py
10  |   |-- graphx_fraud_network.py
11  |   |-- evaluation_visualization.py
12  |-- outputs/
13  |   |-- metrics/
14  |   |-- predictions/
15  |-- grafana/
16  |   |-- fraud_detection_dashboard.json
17  |   |-- data/
18  |-- azure/
19  |   |-- arm-template.json
20  |   |-- databricks_config.py
21  |-- screenshots/
22  |-- docs/
23  |-- README.md

```

B Execution Commands

Listing 8: Commands to run the pipeline

```

1 # 1. Generate test data (optional)
2 python src/generate_sample_data.py
3
4 # 2. Spark SQL Analytics

```

```
5 spark-submit src/spark_sql_analytics.py
6
7 # 3. MLlib Model Training
8 spark-submit src/mllib_fraud_model.py
9
10 # 4. Streaming Simulation
11 spark-submit src/streaming_fraud_detection.py
12
13 # 5. Prepare Grafana Data
14 python src/prepare_grafana_data.py
```